A TWO-LEVELED WEB SERVICE PATH RE-PLANNING TECHNIQUE

Shih-Chien Chou and Chih-Yang Chiang

Department of Computer Science and Information Engineering
National Dong Hwa University, Taiwan
scchou@mail.ndhu.edu.tw

ABSTRACT

Web service paths can accomplish customer requirements. During the execution of a web service path, violations of service level agreements (SLAs) will cause the path to be re-planed (healed). Existing re-planning techniques generally suffer from shortcomings of ignoring the effect of requirement change. This paper proposes a two-leveled path re-planning technique (TLPRP), which offers the following features: (a) TLPRP is composed of both meta and physical levels. The meta level re-planning senses environment parameters (e.g., requirement change and analysis/design errors) and re-plan the affected component paths produced by system design. (b) The physical level re-planning algorithm possesses the ability of web service path composition. (c) The re-planning algorithm embeds an access control policy that computes a successful possibility for every path to facilitate avoiding execution failure caused by web service access failure.

KEYWORDS

Web service, Re-planning, Service level agreement (SLA), Meta level

1. INTRODUCTION

When a requester intends to accomplish a function using web services, he selects one or a composite of web services. If a composite of web services is selected, the invocation order forms a *web service path*. To execute a path, a mechanism such as BPEL [1] can be used. If problems occur during path execution, a re-planning process can heal the path. Before discussing re-planning, we clarify the following concepts.

a. Web services are generally *atomic* (i.e., they offer few and cohesive functions) because: (a) a non-atomic web service is expected to be more expensive because of more functions offered and (b) requesters may only need parts of the functions in a non-atomic web service, which causes them to invoke atomic ones.
b. Customers (i.e., requesters) invoke web services to accomplish their requirements. A customer requirement is generally complicated that should be accomplished by a composite of web services instead of an atomic one.

Item *a* implies that a requirement is difficult to be accomplished by an atomic web service. Item *b* implies that most web service researches ignore software engineering process. In our opinion, software engineering process and web service application should cooperate. A software engineering process contains the phases of requirement capturing/analysis, system design, implementation, testing, and so on. We think that web services cannot be applied earlier than system design because: (a) a requirement is generally complicated that cannot be accomplished by an atomic web service and (b) system design generally identifies atomic software components which are suitable to be accomplished by web services. If system design produces complicated components, decomposing them can solve the problem. According to the above description, we think that software engineering and web service application are dependent and the application of web services can follow the phase of system design.

As mentioned above, web service paths accomplish customer requirements. Generally, a path accomplishes one or more closely related requirements. During path execution, a web service may fail according to quality of service (QoS) or functionality dissatisfaction. In any case, the path should be healed (re-planned). Many re-planning techniques have been proposed. Some of them self-heal an ill path [2-4], some enhance the failure handing ability of BPEL for path healing [2, 5], and some use redundant web services [6-9]. Our survey reveals the shortcomings of the existing re-planning techniques as follows.

a. Most techniques operate on the web service level but ignore customers. Since web service paths accomplish customer requirements, changing customer requirements will cause the affected web service paths to become incorrect.
b. Most techniques ignore the correctness of functionality.
c. If re-planning is based on self-healing or web service replacement, unpredicted failures may invalidate the technique because no solution is offered for the failures.
d. If customers change requirements or an unpredicted failure occurs, the affected web service paths should be re-composed. Most techniques fail to take this point into consideration.
e. Web service access failures may cause path execution failure [2]. Most techniques fail to take access control into consideration.

We develop a new technique to overcome the shortcomings. The design philosophy of our re-planning technique is described in the following paragraphs.

As mentioned, system design identifies atomic software components. One or some closely related requirements can be accomplished by a composite of software components. The components should be executed following an order, which causes the components to form a *component path*. An element in a component path can be accomplished by a web service (complicated components can be decomposed). We call the activities before and after applying web services as the *meta level* activities and the *physical level* ones, respectively. After identifying component paths, our technique identifies web service paths for each component path following the procedure: (a) one or more web services are selected for each component in which the selected ones should fulfill the QoS criteria, (b) an access control policy is applied to filter out web services that cannot be invoked by the requester, (c) a composition algorithm is applied to compose *multiple web service paths* for the component path, (d) software engineers select one or more composed paths in which one is selected to execute and the others are *spare paths*, and (e) if violations of the service level agreement (SLAs) [10] are detected during path execution, the path is re-planned. The re-planning can be achieved using web services or sub-paths of the spare paths. This explains why we compose multiple paths. If spare paths cannot heal the ill path (e.g., the component path is changed because of requirement change), the re-planning algorithm is applied (the algorithm is in fact a path composition algorithm). According to the above procedure, our path re-planning technique offers the following features.

a. The re-planning technique is composed of both meta and physical levels. When meta level re-planning is required (e.g., when customers change requirements), the affected web service paths become incorrect and will be suspended immediately. After that, the affected component paths are re-designed and the corresponding web service paths are re-composed
b. If the meta level re-planning does not occur, the physical level re-planning takes action when a

violation of the physical level SLAs is detected. Since our path composition algorithm composes multiple paths, web services and even sub-paths of the spare paths can be selected to replace the failed web service. If the replacement cannot heal the path, the re-planning algorithm is initiated.

c. The re-planning algorithm embeds an access control policy to compute a successful possibility for every path. The possibilities facilitate avoiding path execution failure caused by web service access failure.

This paper proposes our path re-planning technique. It is composed of the meta and the physical levels. It is named TLPRP (two-leveled path re-planning).

2. Related Work

The technique in [5] uses dynamic description logic (DDL) to describe the preconditions and effects of web services. The DDL-described preconditions and effects are transferred into diagnosing processes embedded in a BPEL execution environment. When a web service is executing and a diagnosing process identifies mismatched predictions or effects, the web service will be isolated for path healing. The technique in [11] proposes a heuristic algorithm to evaluate whether the remaining resources (e.g., time and budget) are sufficient for the un-finished sub-path. If the answer is negative, the unfinished sub-path should be re-planned. The technique in [2] predicts seven types of web service failures and proposes solutions to recovery the failures. The recovery solutions are embedded in the BPEL execution environment. The technique in [12] uses semi-Markov model to predict the performance of an executing web service. If the predicted performance failed to fulfill the QoS criteria, the re-planning process is initiated. In this case, the re-planning process and the web service path are executed simultaneously, which reduce the time of re-planning. The technique in [3] proposes that the failure of an executing web service may cause other executing ones to fail because more than one close related web service of a path may be executed in parallel. It uses direct compensation dependency (DCD) and indirect compensation dependency (ICD) to identify the scope of web services affected by a failure. It then reselects web services to replaces those within the scope. The reselection should offer the least compensation. The technique in [4] belongs to the WS-DIAMOND project [13]. It proposes a QoS-driven, connector-based proactive healing architecture. Connectors can intercept the message sent to a web service and add QoS requirements to the message. After the execution of a web service, the QoS values are recorded in a log. The diagnostic engine periodically checks the log. When a possible violation of QoS criteria is detected, the technique initiates a re-configuration mechanism to self-heal the web service path. The technique in [14] compares the exact QoS values with the estimated ones. When a loop of web services is executing, the exact QoS values of the web services are used to evaluate whether the loop may violate the QoS-based SLAs. If the answer is positive, the technique uses an algorithm to identify the path slice (i.e. sub-path) that should be re-planned. The technique in [15] defines the scope of failed web service and embeds the scope in BPEL. When failure occurs, the BPEL engine initiates the healing process for the scope, which selects new web services to replace the failed ones. During the selection, the local and global business rules are used to filter out infeasible web services.

According to the description in section 1, we emphasize the necessity of cooperation between the meta and physical levels of path re-planning. In our survey, we cannot identify a technique that does this.

3. TLPRP

Our re-planning technique TLPRP is depicted in Figure 1. The dotted blocks are the meta and physical level re-planning mechanisms, respectively. Below we explain the figure.

a. The top left block is the early phases of software engineering, including requirement engineering and system design. After system design, cohesive software components are identified. A composite of components solving one or some close related requirements constitute a component path.

b. The bottom left block is the selection and composition process. It selects one or more web services for every component and composes web service paths for component paths. During selection and composition, an access control policy is applied.

c. The bottom block is a path executer.

d. The lower dotted block is the physical level re-planning mechanism. It is composed of a physical re-planning monitor and an algorithm. The monitor detects violations of the physical level SLAs. A violation will trigger the physical re-planning algorithm.

e. The upper dotted block is the meta level re-planning mechanism. It is composed of a meta re-planning monitor and an algorithm. The monitor detects the violates of the meta level SLAs related to requirement change, analysis/design errors, and platform adaption. A violation of the SLAs will trigger the meta re-planning algorithm.

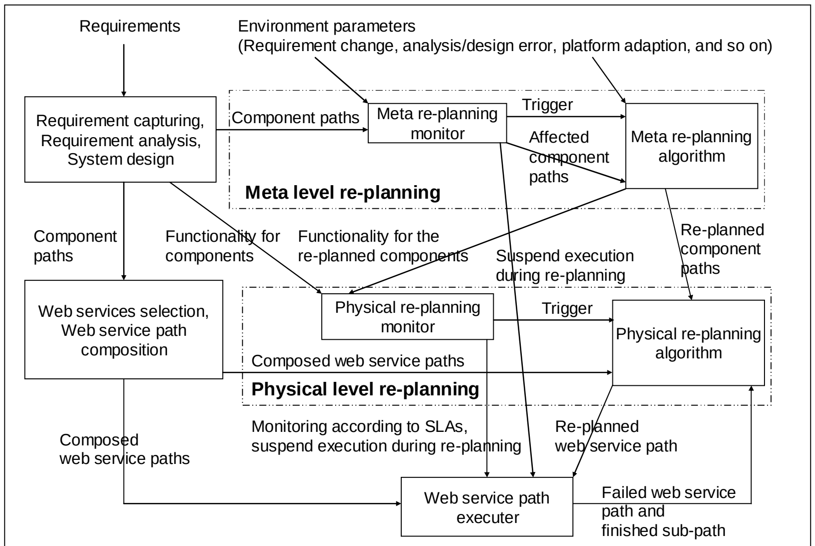

Requirements
Environment parameters
(Requirement change, analysis/design error, platform adaption, and so on)
Requirement capturing, Requirement analysis, System design
Component paths
Meta re-planning monitor
Trigger
Affected component paths
Meta re-planning algorithm
Meta level re-planning
Component paths
Functionality for components
Functionality for the re-planned components
Suspend execution during re-planning
Re-planned component paths
Web services selection, Web service path composition
Physical re-planning monitor
Trigger
Physical re-planning algorithm
Composed web service paths
Physical level re-planning
Monitoring according to SLAs, suspend execution during re-planning
Re-planned web service path
Composed web service paths
Web service path executer
Failed web service path and finished sub-path

Figure 1. TLPRP and the related activities

3.1 Physical Level Re-planning

The physical level re-planning mechanism is composed of a physical re-planning monitor and an algorithm (see Figure 2) as described below.

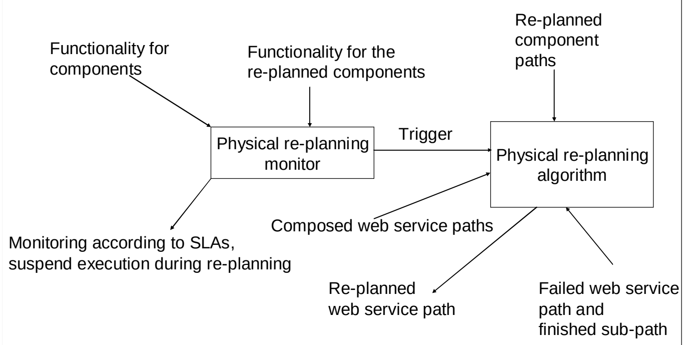

Functionality for components
Functionality for the re-planned components
Re-planned component paths
Physical re-planning monitor
Trigger
Physical re-planning algorithm
Composed web service paths
Monitoring according to SLAs, suspend execution during re-planning
Re-planned web service path
Failed web service path and finished sub-path

Figure 2. Physical level re-planning

Physical re-planning algorithm. As mentioned, the physical re-planning algorithm should possess path composition ability. We thus designed our physical re-planning algorithm as a path composition algorithm. Unlike most composition algorithms, our algorithm *composes multiple web service paths* for a component path. It is for possible replacement during re-planning. When composing web service paths, every path and web service should fulfill the QoS criteria. We suppose the value of every QoS criterion should be large. For a criterion such as *cost* that should be small, the requester should set a maximum value for the criterion to minus the criterion's value. To check QoS, every QoS criterion should be given a limitation. If a QoS criterion's value should be small, we let its value be its maximum value minus its original one. After that, every QoS criterion's value should be at least as large as its limitation. Details of the algorithm are shown in [16]. Below we use the component path in Figure 3 as an example to describe the algorithm.

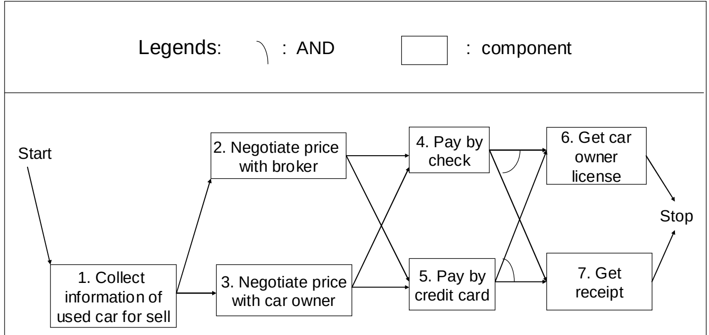
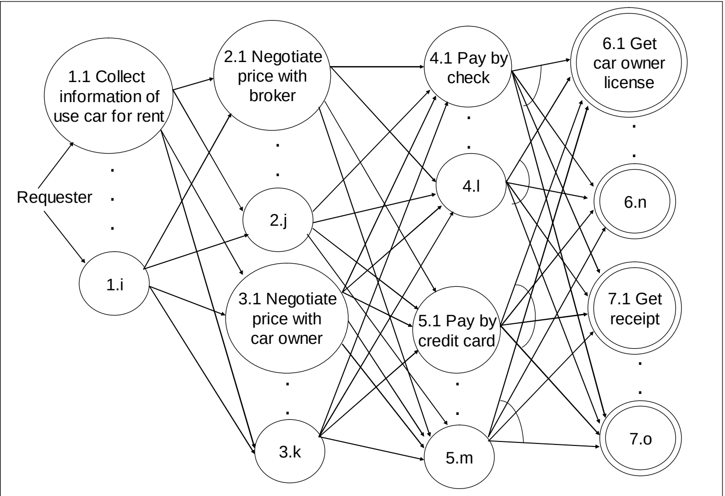


Figure 3. The component path for the requirement "buy a used car"



1.1 Collect information of use car for rent
2.1 Negotiate price with broker
4.1 Pay by check
6.1 Get car owner license
Requester
4.l
6.n
2.j
1.i
3.1 Negotiate price with car owner
5.1 Pay by credit card
7.1 Get receipt
3.k
5.m
7.o

Figure 4. The *WSIG* for the component path "Buy a used car"

When composing web service paths for the component path in Figure 3, web services for every component in the component path are first selected. Suppose the selection produces a web service invocation graph (*WSIG*) in Figure 4, in which one component in Figure 3 can be accomplished by multiple web services offering the same function. Double-circles in Figure 4 are the last web services that should be finished for a path. Logical relationships such as AND appearing in the outgoing arrows of a component will also appear in the outgoing arrows of the corresponding web services in a *WSIG* (please check Figures 3 and 4 to confirm this).

After selection, our *two-leveled access control policy* is applied. The upper level checks whether the requester possesses the *attributes* and *credentials* required by a web service to filter out web services that cannot be invoked by a requester. The lower level uses the *credit level numbers* of web services (larger numbers correspond to more believable web services) and *security level numbers* of arguments (larger numbers corresponds to more sensitive information) to evaluate the possibility of a requester that can invoke a web service using the formula below, in which: (a) $POSI_{succ}$ is the possibility of a requester to successfully invoke a web service *ws*, (b) *k* is "$ws_{cln} - max_{sln}$", in which ws_{cln} is the credit level number of *ws* and max_{sln} is the maximum security level numbers of the arguments sent to *ws*, and (c) *n* is a threshold to facilitate computing $POSI_{succ}$.

$$\begin{aligned} &POSI_{succ} = 1, && \text{if } k \geq n \\ &POSI_{succ} = (k+n)/2n, && \text{if } k \text{ is between } -n \text{ and } (n-1) \\ &POSI_{succ} = 0, && \text{if } k < -n \end{aligned} \qquad (1)$$

Suppose Figure 5 is the *WSIG* after applying the upper level access control policy. To identify web service paths from the *WSIG*, the algorithm transfers it into a web service invocation tree (*WSIT*), because a leaf in a tree exactly belongs to one path. To transfer a *WSIG* into a *WSIT*, the operation *TrTree* listed below can be applied. After applying *TrTree*, Figure 5 will be transferred into Figure 6. Note that we skip the case of loops (i.e., we only handle acyclic component paths). We let the case be a future work.

TrTree. Repeat

Identify a node ND_i in *WSIG* with *N* incoming arrows in which $N > 1$

```
    Duplicate $N$ sub-*WSIG* rooted at $ND_i$
    Let every incoming arrow point to the root node of one duplicated sub-graph
Until every node is pointed by only one incoming arrow
```

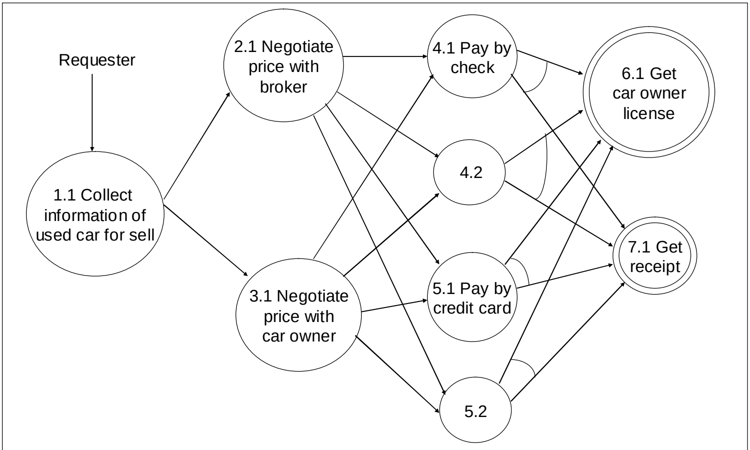

Requester
1.1 Collect information of used car for sell
2.1 Negotiate price with broker
3.1 Negotiate price with car owner
4.1 Pay by check
4.2
5.1 Pay by credit card
5.2
6.1 Get car owner license
7.1 Get receipt

Figure 5. The *WSIG* for the component path "buy a used car" after applying the upper level access control policy in which many web services have been filtered out

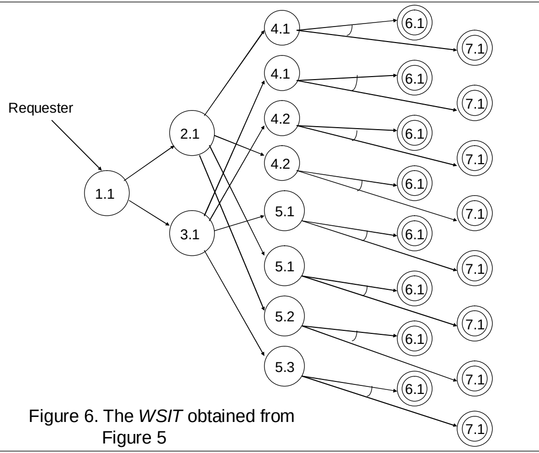


Figure 6. The *WSIT* obtained from Figure 5

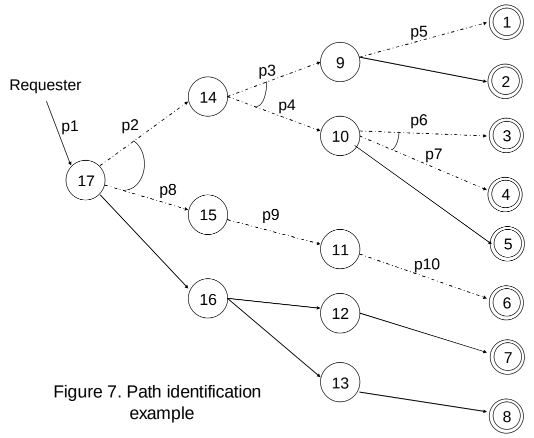


Figure 7. Path identification example

After constructing a *WSIT*, the algorithm identifies web service paths by backtracking the *WSIT* starting from its leaves. We use Figure 7 to explain the path identification process. Suppose node 1 is visited first. Then, the backtracking starts from node 1 up to 14. Since an AND covers two outgoing arrows of node 14, the backtracking rewinds into a forward depth first search process to identify node 10, 3, and 4. After that, the backtracking restarts from node 14 and backs to 17. The AND covering node 17 starts another depth first search. After that, the path containing nodes connected by dotted-lines in Figure 7 is identified.

Having identified all paths from a *WSIT*, our lower level access control policy evaluates the possibility of a requester that can invoke a web service using Formula 1. For example, *p2* in Figure 7 is the possibility of a requester that can invoke web service 14. After the evaluation, the possibility of successfully finishing a path can be computed using $\prod_i p_i$. We call the possibility *SUCCPTH*. In addition to *SUCCPTH*, our algorithm also evaluates the value of QoS criteria of a path. We first compute the value of each QoS criterion for the path. For example, the cost of a path is the summation of the web services' costs in the path. After the value of every criterion has been computed, the expression $\sum_i (Qp_i * Wp_i)$ computes the QoS value *QoSPTH* for the path, in which Qp_i is the ith QoS criterion's value, Wp_i is the weight of Qp_i, and $\sum_i Wp_i = 1$. After that, the overall value of a path *PTHVAL* is computed using the expression "*SUCCPTH * w1 + QoSPTH * w2*", in which *w1* and *w2* are respectively the weights of the two values and their summation is one. By referring to *PTHVAL*s of the paths, the requester selects a path to execute and more or less others for spare paths.

Physical re-planning monitor. The physical re-planning monitor monitors web service path execution using physical level SLAs obtained from web service functionality and QoS criteria. Since we do not intend to design a QoS model, we only monitor the important QoS criteria of *cost*, *time*, *reliability*, and *availability*. A violation of the functional SLAs will cause the web service to be replaced. A violation of the QoS-based SLAs will cause the monitor to check whether the left resources are sufficient for the unfinished sub-path. The SLAs and the rules to monitor the execution of a web service path are described below.

a. The monitoring of web service function. It is impossible to know how a web service executes. We thus check the arguments and return data of a web service for the monitoring. The following operators are used for the checking: (a) logic operators such as AND, OR, NOT, ForAll, and ThereExist, (b) arithmetic operators such as +, -, *, and /, (d) comparison operators such as >, >=, =, <>, <=, and <, (d) set operators such as BelongTo, =, <>, -, Include, Union, and Intersection, and (e) condition operators such as "*if … then … else*". For example, if a web service sorts the data in array *a*, then the functional SLAs can be "ForAll i, a[i] >= a[i+1]". As another example, suppose a web service computes the discount rate of a customer according to the following policies: (a) if the customer is a member with a class of *Gold* or *Platinum*, the customer consumes at least USD 3000 last month, and the customer is neither a vice president nor a president, then the discount rate should be between 0.75 and 0.8, (b) if the first two conditions are the same as the previous policy but the customer is a vice president or a president, then the discount rate should be between 0.65 and 0.7, (c) if one or more of the first two conditions in item *a* are false and the customer is neither a vice president nor a president, then the discount rate should be between 0.9 and 0.95, and (d) if one or more of the first two conditions in item *a* are false but the customer is a vice president or a president, then the discount rate should be between 0.8 and 0.85. In this case, arguments sent to the web service includes *MCLS* (member class), *CAMT* (consumption amount of the last month), and *TITLE*. Moreover, the web service returns a discount rate *DISR*. Under the policies listed above, the functional SLA can be described below.

```
if (MCLS BelongTo {Gold, Platinum}) AND (CAMT >= 3000) then
    if (NOT(TITLE BelongTo {vice president, president})) then
        0.75 <= DISR <= 0.8;
    else
        0.65 <= DISR <= 0.7;
    end if;
else if (NOT(TITLE BelongTo {vice president, president})) then
        0.9 <= DISR <= 0.95;
    else
        0.8 <= DISR <= 0.85;
    end if;
end if;
```

To interpret the SLAs, the monitor is designed as a simple parser. If the monitor identifies violations of functional SLAs, the physical re-planning algorithm is triggered and the executing web service path is suspended. The following three possible activities may be used for the re-planning: (a) suggest spare web services to the requester, (b) suggest spare sub-paths to the requester, and (c) execute the composition algorithm to identify a new web service or web service sub-path.

b. The monitoring of QoS criteria. When a web service is invoked, a timeout counter is used to monitor reliability and availability. When timeout occurs and the execution of the web service is not finished, the QoS-based SLAs according to reliability or availability is violated and the three options described near the end of the previous paragraph are applied. If a web service returns in time and the function is correct, the monitor checks whether the SLAs according to cost or time are violated. The checking can be achieved by comparing the actual cost/time spent with that promised by a web service. If a violation of the SLAs occurs, the monitor does not care the finished web service because its function is correct. Instead, the monitor checks whether the left budget/time is sufficient for the unfinished sub-path. If the checking passes, the sub-path executes normally. Otherwise, the physical re-planning algorithm will be triggered. The following possible activities may be taken: (a) suggest to the requesters the spare sub-paths that can finish using the left budget and time and (b) execute the composition algorithm to identify a new sub-path that can finish using the left budget and time. It is possible that no sub-path can replace the unfinished sub-path because of insufficient resources. In this case, the physical re-planning monitor will notify the requester.

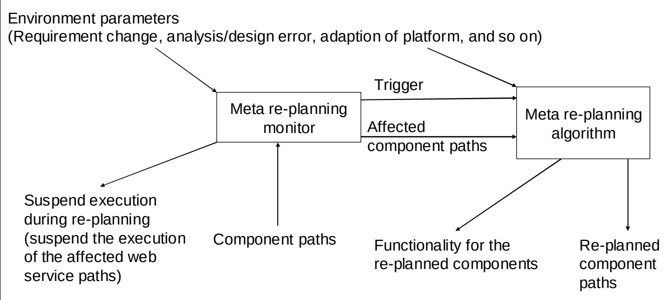

Environment parameters
(Requirement change, analysis/design error, adaption of platform, and so on)
Trigger
Meta re-planning monitor
Affected component paths
Meta re-planning algorithm
Suspend execution during re-planning (suspend the execution of the affected web service paths)
Component paths
Functionality for the re-planned components
Re-planned component paths

Figure 8. Meta level re-planning

3.2 Meta Level Re-planning Mechanism

The meta level re-planning will be activated when *environment parameters* are detected, in which environment parameters include all factors that will cause the software to change. Example environment parameters are *requirement change, analysis/design errors, and adaptation of platform.* The physical level re-planning is enough if no environment parameter occurs. However, customers change requirements frequently, errors on requirement analysis or design are identified occasionally, and adaptation of platform is needed sometimes. Either of the cases will produce environment parameters. An environment parameter will change one or more component paths produced by system design, which will in turn change the corresponding web service paths. In this section, we describe the management of component path re-planning according to environment parameters. That is, this section describes the meta level re-planning.

The meta level re-planning mechanism is composed of a meta re-planning monitor and an algorithm, as shown in Figure 8. The meta re-planning monitor senses environment parameters. As long as the monitor senses any environment parameter, it first identifies the component paths affected by the parameters. It then suspends the executing web service paths implementing the affected component paths. The suspension is necessary because the affected component paths as well as the corresponding web service paths should be changed (i.e., the affected paths become incorrect). After the suspension, the monitor triggers the meta re-planning algorithm to plan new component paths according the environment parameters sensed by the monitor. The above description reveals that the meta level SLAs used in the meta re-planning monitor include customer requirements, the results of requirement analysis and system design, the platform that executes the software, and so on. Any change of the above mentioned items results in a violation of the meta level SLAs (i.e., produces an environment parameter). Since the meta level monitor should sense environment parameters and identify the affected component paths, the monitor is difficult to automate. Therefore, the functions of the meta level monitor in TLPRP are performed by software engineers.

When a violation of the meta level SLAs occurs, the meta re-planning algorithm is triggered. The algorithm re-plans the affected component paths. The re-planning algorithm is actually a software maintenance process and performed by software engineers. During the re-planning, software engineers re-capture and re-analyze customer requirements, and re-design the affected component paths. The engineers will take the original component paths as a reference during the re-planning.

Generally, the engineers should change the original component paths at least as possible to reduce the effort of the physical level re-planning (see the discussion in section 3.3).

According to the above description, the meta re-planning algorithm and the software engineering process can be the same. A typical software engineering process is composed of requirement capturing/analysis, system design, implementation, testing, and maintenance. The phases of implementation and testing in web service applications are replaced by web service selection, composition, and execution. Below we discuss the requirement and design phases in the meta level re-planning of TLPRP.

Requirement capturing and analysis identify *what* customers want. System design focuses on *how* to achieve customer requirements. In general, a requirement can be solved by more than one solution. For example, the requirement “travel around the world” can be solved using the solutions “take airplanes and stay in luxury hotels for the nights”, “take ships and stay in the rooms of the ships for the nights”, and “take ships or trains and stay in the rooms of the ships or in camps for the nights”. Among the solutions of a requirement, which one will be selected? What dominate the selection? In fact, the identification of solutions for requirements and the selection of proper ones are classical problems. Many comprehensive researches can be identified. We do not intend to propose a new technique to identify and select solutions for requirements. Instead, we use the existing research results to implement our meta re-planning algorithm.

As described above, the meta re-planning algorithm and the software engineering process can be the same. In the meta re-planning algorithm, we use existing techniques such as meetings or interacting with customers to identify customer requirements. We then use the use case diagram of UML [17] to represent the requirements. After that, we use existing system design process and metrics to design component paths. No matter what design process is used, our purpose is identifying components that meet design metrics such as high cohesion and low coupling. The design metrics, which are regarded as *meta level QoS criteria* in this paper can be identified from much published material. When selecting the criteria, we consider the following three factors: (a) localizing change effects, (b) fulfilling the budget and time requirements of customers, and (c) requiring that a component can be accomplished by a web service. From the first factor, we select the meta level QoS criteria “cohesion” and “coupling”. From the second, we select “cost” and “time”. From the third, we select “size” and “cohesion” (a cohesive component of proper size has more chance to be accomplished by a web service). Note that the physical level QoS criteria are generally offered by web service providers whereas the meta level ones are obtained from estimation.

With the meta level QoS criteria and the solutions identified by applying a system design process, the meta re-planning algorithm can select one optimal solution according to the meta level QoS criteria (here *a solution is a component path*). The re-planned component path is then sent to the physical re-planning algorithm for web service path re-planning. Remember that the physical re-planning algorithm composes multiple paths for possible replacement. On the contrary, the meta re-planning algorithm selects only one component path. The rationales are listed below:

a. In the physical level, web services for a component offers the same function. When a re-planning is required, using another one to replace the failed one is meaningful.
b. In the meta level, re-planning corresponds to changing solutions (i.e., changing component paths). In this situation, planning or re-planning multiple component paths for a requirement is meaningless.

3.3 Interaction Between the Two Levels of Re-planning

According to the description in sections 3.1 and 3.2, the re-planning technique of the meta level and that of the physical level are different, as described below:

a. The physical level re-planning is primarily achieved by monitors and algorithms executed by computers. On the contrary, the meta level re-planning is achieved by software engineers and customers.
b. The physical level SLAs are obtained from web service functionality and QoS criteria whereas the meta level SLAs are environment parameters produced by customers or software engineers. Moreover, the physical level monitor *proactively* monitors whether physical SLAs are violated whereas the meta level monitor *reacts* to environment parameters.
c. The QoS criteria of the physical level are generally offered by web service providers whereas those of the meta level are obtained by estimation.

Although differences exist between the two levels of re-planning, they should cooperate closely. In the normal case, component paths are identified after system design. The corresponding web service paths are then composed and executed. If violations of the physical level SLAs are detected on a web service path, the physical level re-planning is activated to heal the path. The re-planning and execution continue until every web service paths finishes execution.

However, as we have mentioned above, environment parameters (including requirement change, analysis or design errors, adaptation of platform, etc.) may occur anytime. Since the physical level re-planning mechanism cannot detect environment parameters, incorrect web service paths may continue execution when environment parameters occur. Continue executing incorrect web service paths wastes resources to do meaningless work. To avoid the meaningless execution when environment parameters occur, the meta level re-planning is needed. From a certain viewpoint, *the meta level re-planning plays the role of a forecaster* for web service paths. That is, when the meta level monitor senses environment parameters, it identifies the affected component paths and suspends their corresponding web service paths. The meta level monitor then trigger the meta re-planning algorithm to re-plan new component paths. The new component paths are then sent to the physical re-planning mechanism to re-plan new web service paths. When re-planning new component paths, the meta re-planning algorithm must change at least components as possible. We use Figure 9 to explain the consideration.

Suppose the component path in Figure 9 is affected by an environment parameter. Also suppose the meta re-planning algorithm does not change components *1* through *i*. It also does not change components *n-1*, and *n*. Then, the selection process and access control policy need not be applied to the unchanged components. Of the most important, suppose the corresponding web service path has been executed to web service *k*. Then, if $k \leq i$, the finished web service sub-path needs not be re-executed. If $k \geq i$, the finished sub-path starts from web service *1* down to *i* needs not be re-executed. The above consideration is important because the change of web service *j* will cause all the finished web services after *j* to re-execute. The rationale is that different arguments may be sent to the finished web services according to the change of web service *j*.

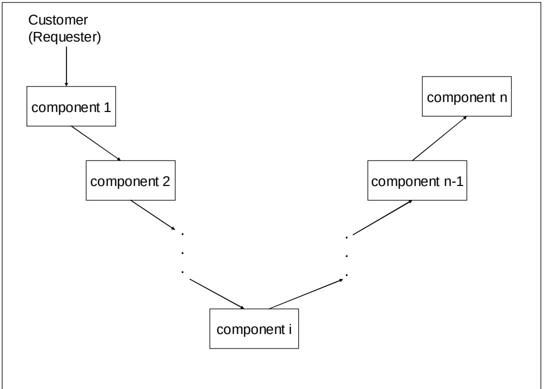


Figure 9. An example component path

4. Example

We use the requirement “buy a used car” as an example to explain the used of TLPRP. The component path of the requirement is shown in Figure 3. Suppose: (a) the corresponding *WSIT* of the component path is shown in Figure 6, (b) the requester selects the dotted-line-linked paths shown in Figure 10, (c) the path composed of the web services 1.1, 2.1, 4.1, 6.1, and 7.1 is selected to execute, and (d) web service 4.1 is executing. Also suppose that violations of the physical level SLAs are detected during the execution of web service 4.1. Then, the physical re-planning algorithm re-plans the web service path using either of the following approach.

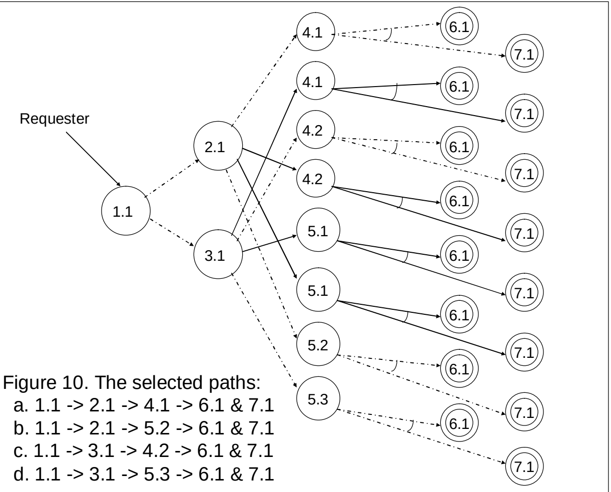


Figure 10. The selected paths:
a. 1.1 -> 2.1 -> 4.1 -> 6.1 & 7.1
b. 1.1 -> 2.1 -> 5.2 -> 6.1 & 7.1
c. 1.1 -> 3.1 -> 4.2 -> 6.1 & 7.1
d. 1.1 -> 3.1 -> 5.3 -> 6.1 & 7.1

a. Replace web service 4.1 by 5.2 and resume the execution from web service 5.2.
b. Replace the sub-path “2.1 -> 4.1 -> 6.1 & 7.1” by “3.1 -> 5.3 -> 6.1 & 7.1” and resume the execution from web service 3.1.
c. Compose sub- web service paths corresponding to the sub- component path starting from components 4 or 5 in Figure 3. Two starting components can be used because execution fails at the web service corresponding to component 4 (i.e. web service 4.1), which is the next one of component 2, and component 2 can be followed by components 4 or 5.

The re-planning describe above occurs on the physical level. Below we describe re-planning on the meta level. Suppose at a certain time, the path composed of the web services 1.1, 2.1, 4.1, 6.1, and 7.1 in Figure 10 is selected to execute and web service 4.1 is executing. Also suppose at this time, the customer decides not to believe any broker and decides to pay by cash or check but not credit card. In this case, the meta level monitor identifies that the executing web service path is affected and suspends the execution immediately. The software engineers then re-design (re-plan) a component path for the changed requirement as shown in Figure 11. The re-planned component path is then transferred to the physical re-plan algorithm. Suppose the algorithm re-plans the *WSIT* and paths in Figure 12. Since component 2 does not appear in Figure 11, the original path selected for execution consisting of 1.1, 2.1, 4.1, 6.1, and 7.1 does not appear in Figure 12. In this case, the requester should select another path to execute. Suppose he selects the path composed of the web services 1.1, 3.1, 4.2, 6.1, and 7.1. Then, the re-planned web service path can be resumed from web service 3.1 because web service 1.1 has been executed before the re-planning.

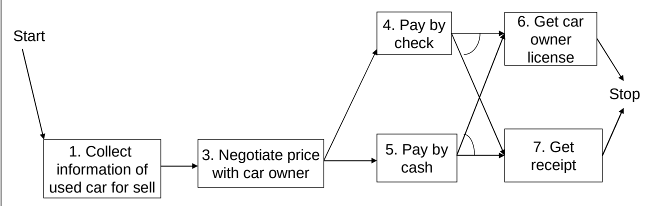

Start
1. Collect information of used car for sell
3. Negotiate price with car owner
4. Pay by check
5. Pay by cash
6. Get car owner license
7. Get receipt
Stop

Figure 11. The re-planned component path for the requirement "buy a used car"

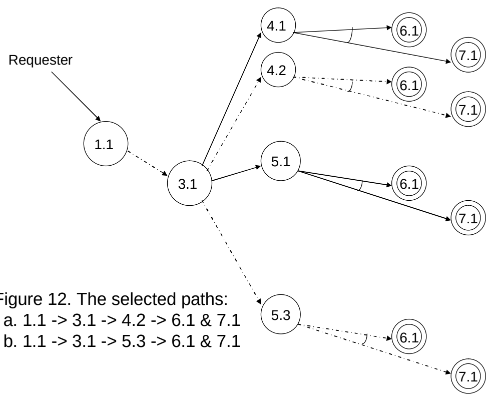


Figure 12. The selected paths:
a. 1.1 -> 3.1 -> 4.2 -> 6.1 & 7.1
b. 1.1 -> 3.1 -> 5.3 -> 6.1 & 7.1

5. EVALUATION

The major contributions of TLPRP include: (a) when an environment parameter is sensed, the meta level re-planning technique re-designs new component paths and causes the corresponding web service paths to be re-planned, (b) the physical re-planning algorithm composes multiple web service paths for possible replacement, and (c) the access control policy facilitates avoiding path execution failure caused by access failure. The first contribution is unsuitable to evaluate by experiments and therefore we evaluated the latter two. We used the component path "Buy a used car" in Figure 3 as an example and selected ten students to attend the experiments. Before the experiments, we required the students to implement 30 web services for each of the component in Figure 3 and gave the web services different QoS criteria values, attributes, and credentials. In addition, each web service was given an initial credit level number. The attributes, credentials, and credit levels were used in the access control policy.

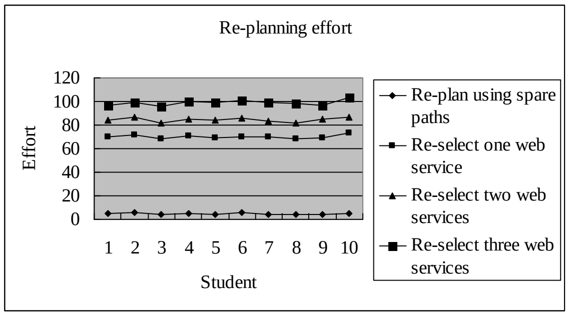


Figure 13. The experiment data of the first experiment

The first experiment estimates re-planning effort. During the experiment, the students were required to *compose five web service paths*. They then selected a path to execute and let the others be spare ones. When executing a path, the students simulated execution failure and performed path re-planning. The re-planning was assisted by the spare paths. We then required the student to estimate the effort needed for the re-planning when spare paths can be used, in which large value corresponds to large effort. We also required the students to simulate the case that spare paths cannot heal the path. The experiment result is shown in Figure 13, which shows the following information:

a. If spare paths can heal an ill path, the re-planning effort is low.
b. If spare paths cannot be used, much effort is needed for physical re-planning. Moreover, as the number of web services needed to be re-planned becomes larger, the re-planned effort will become higher because web service selection needs human assistance.

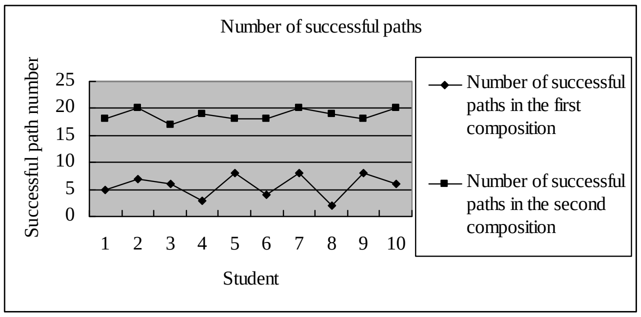


Figure 14. The experiment data of the second experiment

The second experiment estimates the effect of our access control policy. During the experiments, we required the students to: (a) compose 20 paths using our algorithm but skip the access control policy and (b) compose 20 paths without skipping the policy. In this experiment, we bypassed the requirements of QoS. Otherwise, it may be difficult to compose that many paths. After the composition, the students executed the paths and checked the successfulness of the execution (i.e., counted the number of paths not banned by the access control policy). During the execution, we set up access control parameters including attribute and credential requirements for web services, credit levels for web services, and security levels for arguments. Moreover, we bypassed the monitoring of SLAs because this experiment checked the usefulness of the access control policy. The experiment result is shown in Figure 14, which points out the importance of access control policy embedding in a re-planning algorithm.

6. CONCLUSION

Web services can accomplish customer requirements. Since customer requirements are generally complicated and difficult to be accomplished by an atomic web service, system designed should finish before web services can be applied. Generally, system design produces atomic components that can be accomplished by web services. Web service paths can thus be composed by referring to the component paths produced by system design. After composing a web service path, it can be executed. During the execution, if the service level agreements (SLAs) are violated, the path should be re-planed (healed).

Many re-planning techniques have been proposed. They generally suffer from the following shortcomings: (a) most techniques ignore the effect of requirement change, (b) most techniques monitors QoS fulfilling but not functional correctness, (c) many techniques are based on self-healing or web service replacement, which cannot handle unpredicted failures, (d) many techniques do not offers the ability of path composition which is needed when requirement change, and (e) most techniques ignore the importance of monitoring access failure. We do not negate the values of existing techniques. Instead, we intend to develop a new technique. This paper proposes our two-leveled path re-planning technique (TLPRP), which offers the following features.

a. TLPRP is composed of both meta and physical levels. The meta level re-planning senses environment parameters and re-plan the component paths. The re-planning will cause the corresponding the physical level web service paths to be re-planned.
b. The physical level re-planning algorithm is a composition algorithm. In other words, the algorithm offers the ability of path composition.
c. The re-planning algorithm embeds an access control policy to compute a successful possibility for every path. The possibilities facilitate avoiding path execution failure caused by web service access failure.

We set up experiments to evaluate TLPRP. The experiment results show that web service or path replacement reduces re-planning effort substantially and the access control policy facilitates avoiding path execution failure caused by web service access failure.

REFERENCES

[1] Alves, A., Arkin, A., Askary, S., Barreto, C. and Bloch, B., "Web Service Business Execution Language, v. 2.0", available on http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
[2] Subramanian, S., Thiran, P., Narendra, N. C., Mostefaoui, G. K., and Maamir, Z., (2008) "On the Enhancement of BPEL Engines for Self-healing Composite Web Services", International Symposium on Applications and Internet, pp33-39.
[3] Yin, Y., Zhang, B., Zhang, X., and Zhao, Y., (2009) "A Self-healing Composite Web service Model", 2009 IEEE Asia-Pacific Service Computing Conference (IEEE APSCC), pp307-312.
[4] Halima, R. B., Drira, K. and Jmaiel, M., (2007) "A QoS-driven Reconfiguration management System Extending Web Services with Self-healing Properties", 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp339-344.
[5] Han, X., Shi, Z., Niu, W., Lin, F., and Zhang, D., (2009) "An Approach for Diagnosing Unexpected Faults in Web Service Flows", 2009 International Conference on Grid and Cooperative Computing, pp61-66.
[6] Jorge, S., Francisco, P, Marta, P., and Ricardo, J., (2006) "Ws-replication: A Framework for Highly

Available Web Services", 15th International Conference on World Wide Web, pp357-366.
[7] Taher, Y., Benslimane, D., and Fauvet, M., (2006) "Towards an Approach for Web Services Substitution", 10th International Database Engineering and Applications Symposium, pp166-173.
[8] Yu, T. and Lin, K. J., (2005) "Service Selection Algorithms for Web Services with End-to-end QoS Constraints", Journal of Information Systems and e-Business Management, Vol. 3, No. 2, pp103-126.
[9] Guo., H., Huai, J., and Angel, H. L., (2007) "Optimal Configuration for High Available Service Composition", 15th International Conference on Web Services, pp280-287.
[10] Available on http://en.wikipedia.org/wiki/Service_level_agreement
[11] Berbner, R., Spahn, M., Repp, N., Heckmann, O., and Steinmetz, R., (2007) "Dynamic Replanning of Web Service Workflows", 2007 IEEE International Conference on Digital Ecosystems and Technologies, pp211-216.
[12] Dai, Y., Zhu, Z., Li, D., Wang, H., Chen, Y., and Zhang, B., (2009) "Low Cost Mechanism for QoS-aware Re-planning of Composite Service", 2009 International Symposium on Electronic Commerce and Security, pp287-291.
[13] IST, (2006) The Web Services Diagnosability, Monitoring, and Diagnostic Project, available on http://wsdiamond.di.uniti.it
[14] Canfora, G., Penta, M. D., Esposito, R. and Villani, M. L., (2005) "QoS-aware Replanning of Composite Web Services", 2005 IEEE International Conference on Web services.
[15] Ren, W., Chen, G., Shen, H., Zhang, J. B., and Low, C. P., (2008) "Dynamic Self-healing for Service Flows with Semitic Web Services", 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Technology, pp598-604.
[16] Chou S. –C and Jhu, J. –Y., (2010) "Access Control Policy Embedded Composition Algorithm for Web Services", 6th International Conference on Advanced Information Management and Service.
[17] Available on http://www.uml.org/
[18] OASIS, (2003) eXtensible Access Control Markup Language (XACML) Version 1.0. OASIS Standard 18.
[19] W3C, Web Services Description Language (WSDL) 1.1. available on http://www.w3.org/TR/2001/NOTE-wsdl-20010315
[20] Available on http://www.w3.org/Submission/OWL-S
[21] Hwang, S. –Y., Lim, E. –P., Lee, C. –H., and Chen, C. –H., (2008) "Dynamic Web Service Selection for Reliable We Service Composition", IEEE Transactions on Services Computing, Vol. 1, No. 2, pp104-116.
[22] Chen, P. P. W., and Lyu, M. R., (2008) "Dynamic Web Service Composition: A New Approach in Building Reliable Web Service", 22nd International Conference on Advanced Information Networking and Applications, pp20-25.

Authors

Shih-Chien Chou is currently a professor in the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan. His research interests include software engineering, process environment, software reuse, and information flow control.

Chih-Yang Chiang is currently a master student in Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan.